Core Java: Fundamental Classes

Persistent Interactive | Persistent University

Key learning points:

At the end of this module, you will be able to understand :

- Use of fundamental classes from java.lang package
- Object class
- Math class
- Wrapper classes
- Autoboxing and Autounboxing
- String, StringBuffer classes
- Date and Calendar classes
- Java.time API

Object class

- Every class that we create extends the class Object by default. That is the Object class is at the highest of the hierarchy. This facilitates to pass an object of any class to be passed as an argument to methods.
- Some of the methods in this class are:

Method name	Description
equals(Object ref)	returns true if both objects are equal
finalize( )	method called when an object's memory is destroyed
getClass( )	returns class to which the object belongs

Object class

Method name	Description
hashCode( )	returns the hashcode of the class object
toString()	return the string equivalent of the object name
notify( )	method to give message to a synchronized methods
notifyAll( )	method to give message to all synchronized methods
wait()	suspends a thread for a while
wait(...)	suspends a thread for specified time in seconds

The Math Class

- Support common mathematical functions.
- Can not be instantiated.
- Two constants :
- Rounding Functions

```
Math.E
Math.PI
```

```
static type abs(type t)
static type min(type t)
static type max(type t)   //type: int/long/float/double
static double ciel(double d)
static double floor(double d)
static type round(type t)   //type: float/double
```

The Math Class…

- Exponential Functions

```
static double pow(double d1, double d2)
static double exp(double d)
static double log(double d)
static double sqrt(double d)
```

- Trigonometric Functions

```
static double sin(double d)
static double cos(double d)
static double tan(double d)
static double toRadian(double degrees)
static double toDegrees(double redians)
```

- Pseudorandom Number Generator

```
static double random()
```

The Wrapper Classes

- Are used to manipulate primitive values as objects.
- Are final.
- Objects of wrapper classes are immutable.

Common Wrapper Class Constructors

WrapperType(type v)

WrapperType(String str)

Converting Primitive Values to Wrapper Objects

Converting Strings to Wrapper Objects

```
Character charObj1 = new Character('\n');
Boolean boolObj1 = new Boolean(true);
Integer intObj1 = new Integer (2005);
```

```
Boolean boolObj2 = new Boolean("TruE");
Boolean boolObj3 = new Boolean("PP"); //false
Integer intObj2 = new Integer("123");
```

Converting Strings to Wrapper Objects

static WrapperType valueOf (String s)

- returns the wrapper object corresponding to the primitive value represented by the String object.
- throws a NumberFormatException if the string is not a valid number.

Character class does not define this method.

```
Boolean boolObj4 = Boolean.valueOf(“false”);
Integer intObj3 = Integer.valueOf(“1999”);
```

Converting Wrapper Objects to Strings

- returns the string object containing the string representation of the primitive value in the wrapper object.

```
String charStr = charObj1.toString();    // “\n”

String boolStr = boolObj2.toString();   // true”

String intStr = intObj1.toString();  // “2005”
```

Converting Primitive Values to Strings

static String toString(type v)

- returns the string corresponding to the primitive value of type passed as argument.

```
String charStr2 = Character.toString(‘\n’);  // “\n”

String boolStr2 = Boolean.toString(true);  // “true”

String intStr2 = Integer.toString(2005);     // “2005”
```

Converting Wrapper Objects to Primitive Values

type typeValue()

- returns the primitive value in the wrapper object.

```
char c = charObj1.charValue();          //'\n'

boolean b = boolObj2.booleanValue(); //true

int i = intObj1.intValue();             //2005
```

Wrapper Comparison

int compareTo(WrapperType obj2)

```
int n = obj1. compareTo(WrapperType obj2)
        n <0        : obj1 less than obj2
        n >0        : obj1 greater than obj2
        n == 0      : obj1 equals to obj2
        Character charObj2 = new Character('a');
        int result1 = charObj1.compareTo(charObj2);
        // <0
        int result2 = intObj1.compareTo(intObj3);
        // >0
        int result3 = boolObj1.compareTo(charObj2);

        //ClassCastException
```

Numeric Wrapper Classes

<wrapper class name>.MIN_VALUE
<wrapper class name>.MAX_VALUE

```
byte minByte = Byte.MIN_VALUE; //-128
```

Converting any Numeric Wrapper Object to any Numeric Type

byte	**byteValue()**
short	**shortValue()**
int	**intValue()**
long	**longValue()**
float	**floatValue()**
double	**doubleValue()**

```
int i = intObj1.intValue(); //2005
```

Numeric Wrapper Classes…

- Converting Strings to Numeric Values

  type parseType(String s)

- Converting Integer Values to Strings in different Notations

```
byte value1 = Byte.parseByte(“16”);
        int value2 = Integer.parseInt(“2005”);
        int value3 = Integer.parseInt(“7UP”);
        //NumberFormatException

static String toBinaryString(int i)

static String toHexString(int i)

static String toOctalString(int i)
```

The Character Class

Constants

```
Character.MIN_VALUE
Character.MAX_VALUE
```

Methods

```
static int getNumericValue(char ch)
static boolean isLowerCase(char ch)
static boolean isUpperCase(char ch)
static boolean isTitleCase(char ch)
static boolean isDigitCase(char ch)
static boolean isLetter(char ch)
static boolean isLetterorDigit(char ch)
static char toUpperCase(char ch)
static char toLowerCase(char ch)
static char toTitleCase(char ch)
```

Auto Boxing & Unboxing

- In Java, some implementations (collections) need data to be present in the form of objects. They can not deal with primitive data members.
- So primitive data needs to be represented in the form of object.
- This is achieved by using the concept of Wrapper classes.
- Representing primitive in the form of object is termed as boxing & converting it back to primitive from reference is termed as unboxing.

Persistent

Auto Boxing

- Auto Boxing is the process where the primitive data member automatically gets converted into its respective wrapper object. No need of explicit conversion by the programmer.

```
Integer intObj1 = new Integer(10); //boxing

Integer intObj2 = 10; //auto boxing
```

Auto Unboxing

- Auto unboxing is the process of automatically extracting the primitive value wrapped inside the object. No need of calling any method explicitly to fetch the primitive value.

```
int num1 = intObj1.intValue(); //unboxing

int num2 = intObj2; //auto unboxing
```

Auto boxing & Unboxing with operators

- Auto boxing & auto unboxing can be performed with comparison operators as well.

```
package com.features;
class UnboxingExample1{
  public static void main(String args[]){
    Integer intObj = 50;
       if(intObj < 70){          //unboxing internally
                 System.out.println(intObj);
        }
  }
}
```

Use of Auto Boxing & Unboxing

```
package com.features;
public class AutoBoxingUnboxingDemo1 {
  public static void main(String[] args) {
                Integer intObj1 = 30; //autoboxing
                Integer intObj2 = 50; //autoboxing
                int result = intObj1 + intObj2; //auto unboxing
and then                                                  // values are added
                System.out.println("Total is: " + result);
   }
}
```

The String Class

- Creating & Initializing Strings
- String Literals is a reference to a String object.

```
String str = “Hello!!!!”;
int len = “Hello!!!!”.length();
// compile time constant expression
String can1 = 7 + “Up”;
String can2 = “7Up”;    //”7Up”
String word = “Up”;
String can4 = 7 + word;
```

String Constructors

String(String s)
String()

String objects can be constructed from arrays of bytes/characters/string buffers.

Reading Characters from a String

```
byte[] bytes = {1, 2, 3, 4};
char[] characters = {‘a’, ‘b’, ‘c’, ‘d’};

char charAt(int index)

void getChar(int srcBegin, int srcEnd, char[] dst, int
dstBegin)

int length()
```

Comparing Strings

```
boolean test = ‘a’<‘b’;          // 0x61<0x62
```

equals()

- Implements String object value equality as two String objects having the same sequence of characters.

```
boolean equals(Object obj)
boolean equalsIgnoreCase(String str2)
```

Comparing Strings

- compareTo()

```
int compareTo(String str2)
int compareTo(Object obj)
```

- Returns a value based on
  - 0 : if current string is equal to str2
  - Less than 0 : if current string is lexicographically less than str2
  - Greater than 0 : if current string is lexicographically greater than to str2

Character Case in a String

String toUpperCase()

String toUpperCase(Locale locale)

String toLowerCase()

String toLowerCase(Locale locale)

String Concatenation

Concatenation of Strings

String Concatenation Operator +

```
String concat(String s)
String billboard = “Just”;
billboard.concat(“lost in space.”); //(1) value not stored
System.out.println(billboard); //(2) ”Just”
billboard = billboard.concat(“ grooving”).concat(“ in heap”); //(3)
System.out.println(billboard); //(4) ”Just grooving in heap”
```

```
String theName = “Uranium”;
String trademark1 = 100 + “%” + theName;
```

Searching for Characters & Substrings

```
int indexOf(int ch)

int indexOf(int ch, int fromIndex)

int indexOf(String str)

int indexOf(String str, int fromIndex)

String replace(char oldChar, char newChar)
```

Searching for Characters & Substrings

- Extracting Substring

```
String trim()

String substring(int startIndex)

String substring(int startIndex, int endIndex )
```

The StringBuffer Class

- implements mutable character strings.
- capacity of the string buffer can also change dynamically.
- The StringBuffer class provides various facilities for manipulating string buffers:
  - constructing string buffers
  - changing, deleting, and reading characters in string buffers
  - constructing strings from string buffers
  - appending, inserting, and deleting in string buffers
  - controlling string buffer capacity

Constructing String Buffers

- Three constructors:
  - **StringBuffer(String s)**
    - Capacity: length of argument string + 16 more characters.
  - **StringBuffer(int length)**
    - Capacity: value of argument length
  - **StringBuffer()**
    - Capacity: 16 characters.

Constructing String Buffers

```
StringBuffer strBuf1 = new StringBuffer(“Java”)
          //Capacity 20

StringBuffer strBuf2 = new StringBuffer(15)
          //Capacity 15
```

Reading & Changing Characters in String Buffers

```
int length()
char charAt(int index)
void setCharAt(int index, char ch)
```

```
StringBuffer strBuf = new StringBuffer(“Javv”);
 // ”Javv” 20

strBuf.setCharAt(strBuf.length()-1,strBuf.charAt(1));
   //”Java”
```

Appending Characters to a String Buffers

StringBuffer append(Object obj)

- obj is converted to a string as if by the static method call String.valueOf(obj), this string is appended to the current string buffer.

StringBuffer append(String str)
StringBuffer append(char[] str)
StringBuffer append(char[] str, int offset, int len)
StringBuffer append(char c)

Appending Characters to a String Buffers

- Primitive value is converted to a string by applying static method call String.valueOf( ), this string is appended to the current string buffer.

```
StringBuffer append(boolean b)
StringBuffer append(int i)
StringBuffer append(long l)
StringBuffer append(float f)
StringBuffer append(double d)
```

Persistent

Inserting & Deleting Characters in a String Buffers

```
StringBuffer insert(int offset, Object obj)
StringBuffer insert(int offset, String str)
StringBuffer insert(int offset, char[] str)
StringBuffer insert(int offset, char c)
StringBuffer insert(int offset, boolean b)
StringBuffer insert(int offset, int i)
StringBuffer insert(int offset, long l)
StringBuffer insert(int offset, double d)
```

Inserting & Deleting Characters in a String Buffers

```
StringBuffer deleteCharAt(int index)
StringBuffer delete(int start, int end )
StringBuffer reverse()
```

```
StringBuffer buffer = new StringBuffer(“banana split”);
    buffer.delete(4,12);   // “bana”
    buffer.append(42);   // “bana42”
    buffer.insert(4,”na”);   // “banana42”
```

Controlling String Buffer Capacity

```
int capacity()

void ensureCapacity(int minCapacity)

void setLength(int newLength)

buffer.setLength(0);
```

Date class

- Belongs to java.util package.
- Represents the specific instant in time, with millisecond precision.
- Constructors are as follows
  - **Date():** Allocate a Date instance with the current time.
  - **Date(long millisSinceEpoch):** Allocate a Date instance with the given time.

Date class

- **SimpleDateFormat** class is used to control the date/time display format.
- **Date** is legacy class, which does not support internationalization.
- API can be referred to go through various methods.

Calendar class

- The **java.util.calendar** class is an abstract class.
- It provides methods for convertion between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR etc.
- Constants hold specific values related to date. Provides support for internationalization.
- Calendar is a abstract class, and you cannot use the constructor to create an instance. Instead, you use the static method Calendar.getInstance() to instantiate an implementation sub-class.

Summary : Session

With this we have come to an end of our session, where we discussed about ….

- Fundamental classes as Object class, Math class
- Wrapper classes and their usage
- String & StringBuffer classes
- Date & Calendar classes
- Java.time API

Appendix

References

Thank you

Reference Material : Websites & Blogs

- http://www.javatpoint.com/wrapper-class-in-java
- https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html
- http://www.javatpoint.com/autoboxing-and-unboxing
- https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html
- http://www.javatpoint.com/difference-between-string-and-stringbuffer
- https://www.ntu.edu.sg/home/ehchua/programming/java/DateTimeCalendar.html

Reference Material : Books

- **Head First Java**
  - By: Kathy Sierra, Bert Bates
  - Publisher: O’Reilly Media, Inc.
- **Java Complete Reference**
  - By Herbert Schildt

Persistent

Thank you!

Persistent Interactive | Persistent University